

HT32 系列单片机 USB 设备开发工具包

编码：AN0309S

简介

HT32 系列的 USB 设备开发工具包由几个部分组成，包括 USB 设备固件库、类范例和基于 Windows 的 UI 演示。它支持所有具有 USB 设备控制器 Holtek HT32 系列单片机。本应用范例有助于用户熟悉 HT32 系列的 USB 设备开发工具包，创建自己的 USB 应用。类范例代码包含了所有的 USB 传输类型，如控制、中断、批量和等时。本文档提供了以下组件的描述：

- HT32 系列的 USB 设备固件库：外设驱动，USB 核心和应用层
- 人机接口设备（HID）范例：控制和中断传输
- 大容量存储范例：批量传输
- 虚拟 COM 端口范例：中断和批量传输
- USB 视频范例：等时传输

HT32 系列USB设备固件库

本章描述 HT32 系列 USB 设备固件库的结构和应用程序编程接口（API）。HT32 系列 USB 设备固件库为实现以下功能提供了参考：

- HT32 USB 设备控制器的初始化
- USB 总线事件处理
- USB 枚举
- USB 描述符
- USB 标准请求
- USB 类请求
- USB 端点 IN 和 OUT 传输

概述

结构

HT32 USB 设备固件库可以分为三层，硬件和外设驱动器、USB 核心以及应用。图 1 显示了每一层的接口及其交互作用。任何具有 USB 控制器的 HT32 系列单片机只需很少修改，就可重复使用 USB 核心层和应用层。对于大多数类应用，需要修改类和描述符，以适应应用的需求。

USB 核心层和 USB 驱动层在一般情况下不需要修改。

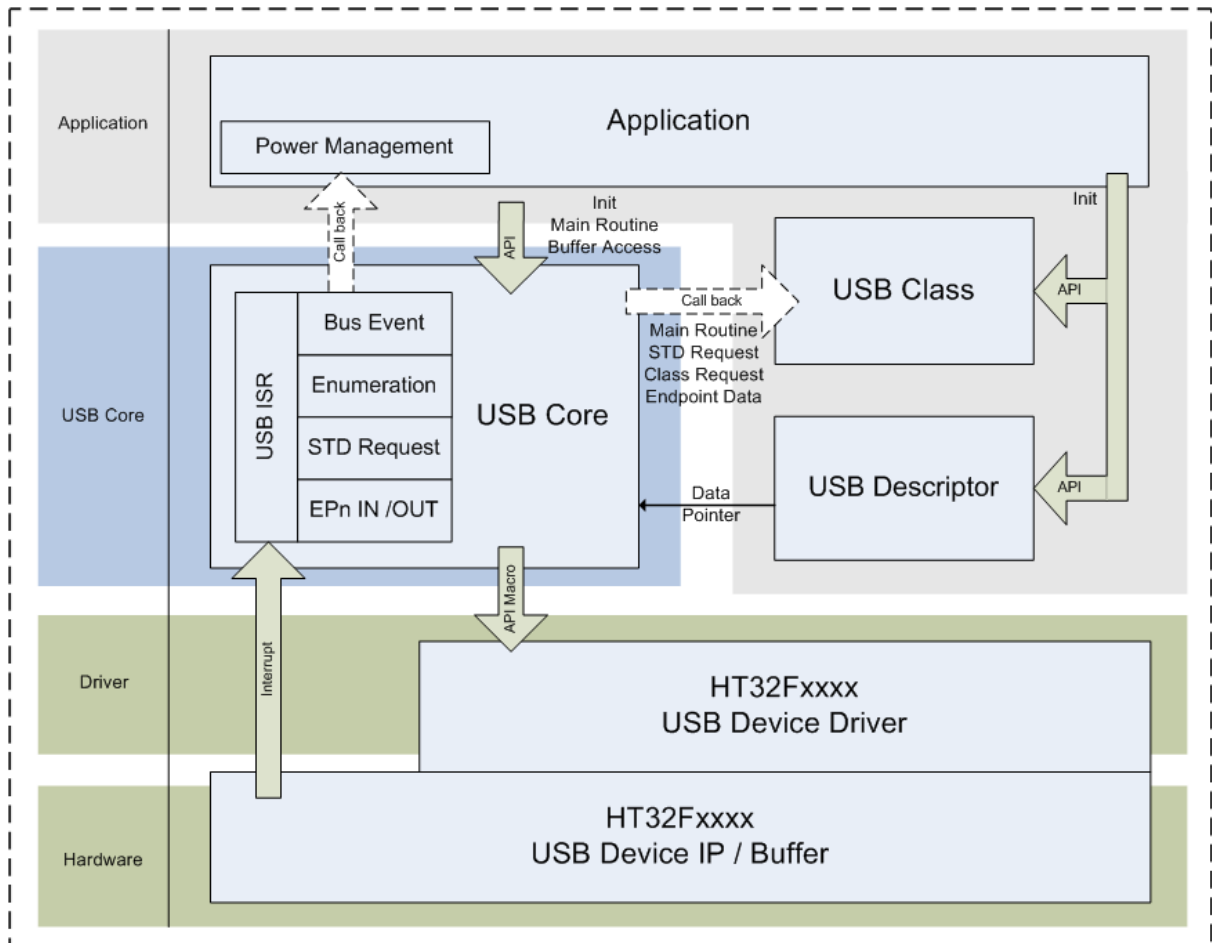


图 1 HT32 USB 设备固件库方框图

每一层的功能：

- **USB 外设驱动**：USB 外设驱动层提供了基本的硬件初始化，寄存器层访问和端点缓冲区访问功能。
- **USB 核心**：USB 核心层负责管理主要的 USB 协议，包括总线事件、枚举和标准请求。它提供了一个中断服务程序来处理所有的 USB 总线活动。
- **应用**：应用层包括 USB 描述符数据、类请求和端点数据处理。USB 类的主要功能设在这一层。

文件组织

图 2 为 HT32 系列单片机的 USB 设备驱动器、USB 核心和类范例代码的文件组织图。请注意，不同类范例代码的类应用层文件可能不同。

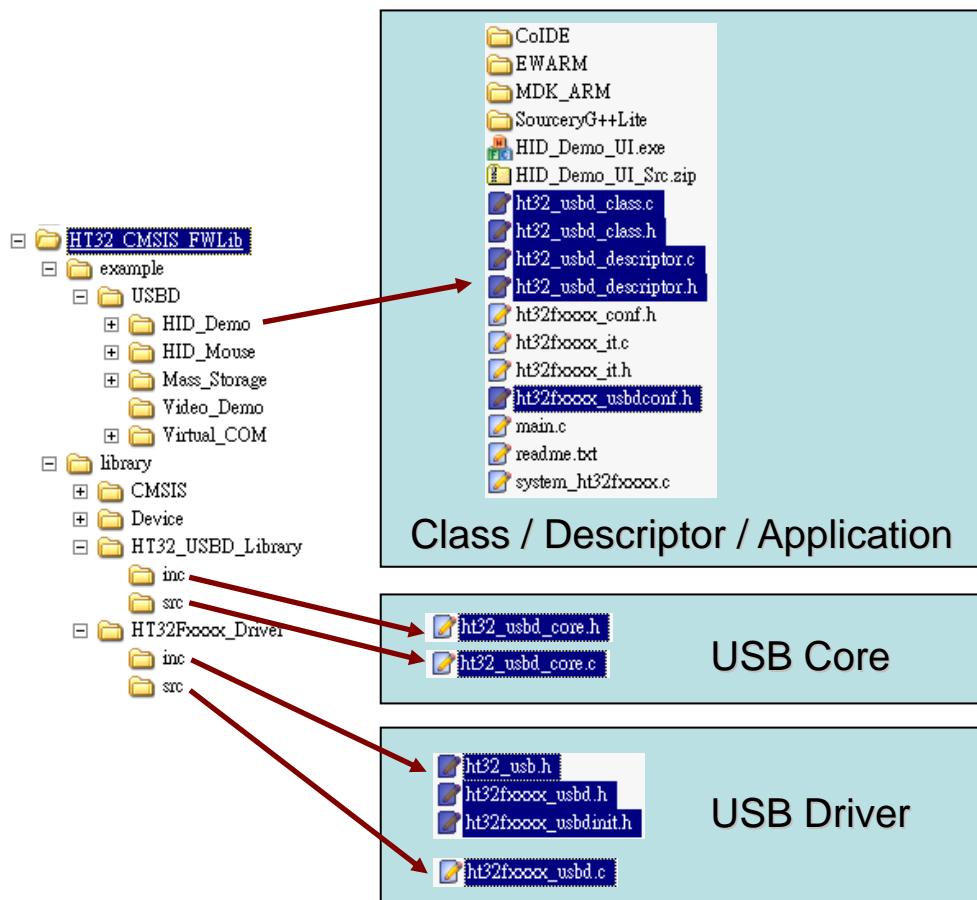


图 2 HT32 USB 设备固件库的文件组织

表 1 显示了每一层的文件：

表 1 HT32 USB 设备固件库的文件列表

层	文件	描述
USB 外设驱动	ht32_usb.h	头文件，提供 HT32 固件库给 API
	ht32fxxx_usbd.h	外设驱动器的头文件
	ht32fxxx_usbdinit.h	从“ht32fxxx_usbdconf.h”文件，检查和处理定义的头文件
	ht32fxxx_usbd.c	外设驱动器的源文件
USB 核心	ht32_usbd_core.h	USB 核心的头文件
	ht32f_usbd_core.c	USB 核心的源文件
应用	ht32f_usbd_class.c	类范例代码的源文件
	ht32f_usbd_class.h	类范例代码的头文件
	ht32_usbd_descriptor.c	描述符的源文件
	ht32_usbd_descriptor.h	描述符的头文件

源代码的命名规则

HT32 USB 设备固件库使用下面的命名规则。明确贴切的命名规则有助于用户更容易理解和修改源代码。

表 2 HT32 USB 设备固件库的命名规则

类型	命名规则/范例	描述
函数名	Prefix_FunctionName (模块名前缀) USBDCore_FunctionName: USB 核心 USBD_FunctionName: USB 驱动 USBDClass_FunctionName: 类应用	所有的函数名添加模块名前缀, 以助于用户识别该函数属于哪个模块。
API 宏	API_USB_NAME (参数) #define API_USB_INIT(driver) (USBD_Init(driver))	USB 核心层使用 API 宏定义作为 USB 驱动层的接口。它为 USB 驱动提供了灵活的函数名。只要在驱动器的头文件中定义特定的 API 宏, USB 驱动器就可以使用任何函数名。
常量定义	#define DEFINE_NAME #define DESC_LEN_HID ((u32)(9))	定义全部大写
枚举类型定义	Prefix_NAME_Enum typedef enum { } USBD_EPTn_Enum;	以模块名前缀开头, 以“Enum”结尾
结构类型定义	Prefix_NAME_TypeDef typedef struct { ... } USBDCore_Info_TypeDef;	以模块名前缀开头, 以“TypeDef”结尾
变量	Pointer: u32 *pDriver;	指针以“p”为前缀
	u8/u16/u32: u32 uPara;	无符号变量以“u”为前缀
	s8/s16/s32: s32 sByteLength;	符号变量以“s”为前缀
	Structure: USBDCore_TypeDef USBCore; Enumeration: USBDCore_Action_Enum Action;	所有的结构或枚举变量都无前缀
全局变量	u8 gInputReportBuffer[64];	全局变量以“g”为前缀

USB 外设驱动

USB 外设驱动包括“ht32_usb.h”、“ht32fxxx_usb.h”、“ht32fxxx_usbdinit.h”和“ht32fxxx_usbd.c”几个文件。下表列出了 USB 外设驱动模块。

表 3 USB 外设驱动模块

模块	描述
ht32_usb.h	USB 核心的 HT32 固件库 API。通过 USB 核心的该文件可以了解外设驱动的文件名。
ht32fxxx_usbd (.h, .c)	外设驱动器提供硬件初始化、寄存器层访问和端点缓冲区访问功能。
ht32fxxx_usbdinit.h	此文件提供了在应用层的“ht32fxxx_usbdconf.h”文件中定义的参数的预处理器检查。

USB 外设驱动使用 API 进行硬件初始化、寄存器层访问、端点缓冲区访问等等。API 可分为两种，全局 API 和端点 API。之后的章节简单介绍了全局 API，端点 API 以及作为 USB 核心层与驱动层接口的 API 宏。更详细的信息，请参考相应设备的编程指南。

全局 API

全局 API 表示 USB 设备控制器共同设置的函数，如电源控制、设备地址和中断操作等。正如我们之前提到的，所有的全局 API 都有“USBD_”前缀和一个便于理解的函数名。下表显示了所有的 API 与其描述。

表 4 USB 外设驱动器的全局 API

API 名称	描述
void USBD_Init(u32 *pDriver);	USB 外设初始化，包括端点的初始值、电源控制和中断屏蔽。
void USBD_PreInit(USBD_Driver_TypeDef *pDriver);	此函数通过应用层的“ht32fxxx_usbdconf.h”文件中的值初始化“USBD_Driver_TypeDef”结构。
void USBD_DeInit(void);	USB 外设取消初始化。
void USBD_PowerOff(void);	进入 USB 设备暂停模式。
void USBD_PowerOn(void);	退出 USB 设备暂停模式。
void USBD_RemoteWakeup(void);	向 USB 主机发送恢复请求进行远程唤醒。
void USBD_ReadSETUPData(u32 *pBuffer);	从 USB 缓冲区读取端点 0 的 SETUP 数据。
void USBD_SetAddress(u32 address);	设置 USB 设备地址。
void USBD_EnableINT(u32 INTFlag);	使能 USB 设备中断。
void USBD_DisableINT(u32 INTFlag);	除能 USB 设备中断。
u32 USBD_GetINT(void);	获取 USB 设备中断标志。
void USBD_ClearINT(u32 INTFlag);	清除 USB 设备中断标志。
USBD_EPTn_Enum USBD_GetEPTnINTNumber(u32 INTFlag);	获取发生中断的有效 USB 端点编号。

端点 API

端点 API 是用来控制端点的初始化、复位、缓冲区访问、总线响应（STALL/NAK/ACK）和中断操作的函数。所有的端点 API 都有“USBD_EPTn”前缀和一个便于理解的函数名。下表显示了所有的 API 与其描述。

表 5 USB 外设驱动器的端点 API

API 名称	描述
void USBD_EPTInit(USBD_EPTn_Enum EPTn, u32 *pDriver);	端点初始化，包括控制和中断寄存器。
void USBD_EPTReset(USBD_EPTn_Enum EPTn);	端点重置到默认状态。
void USBD_EPTEnableINT(USBD_EPTn_Enum EPTn, u32 INTFlag);	使能端点中断。
u32 USBD_EPTGetINT(USBD_EPTn_Enum EPTn);	获取端点中断标志。
void USBD_EPTClearINT(USBD_EPTn_Enum EPTn, u32 INTFlag);	清除端点中断标志。
u32 USBD_EPTGetHalt(USBD_EPTn_Enum EPTn);	获取端点暂停状态。
void USBD_EPTSendSTALL(USBD_EPTn_Enum EPTn);	在指定端点发送 STALL。
void USBD_EPTSetHalt(USBD_EPTn_Enum EPTn);	设置端点暂停状态。
void USBD_EPTClearHalt(USBD_EPTn_Enum EPTn);	清除端点暂停状态。
void USBD_EPTWaitSTALLSent(USBD_EPTn_Enum EPTn);	等待 STALL 发送。
void USBD_EPTClearDTG(USBD_EPTn_Enum EPTn);	清除端点数据触发位。
u32 USBD_EPTGetBuffer0Addr(USBD_EPTn_Enum EPTn);	获取端点缓冲区 0 的地址。
u32 USBD_EPTGetBuffer1Addr(USBD_EPTn_Enum EPTn);	获取端点缓冲区 1 的地址。
u32 USBD_EPTGetBufferLen(USBD_EPTn_Enum EPTn);	获取端点缓冲区长度。
u32 USBD_EPTGetTransferCount(USBD_EPTn_Enum EPTn, USBD_TCR_Enum type);	获取端点传输计数。
u32 USBD_EPTWriteINData(USBD_EPTn_Enum EPTn, u32 *pFrom, u32 len);	写 IN 数据到 USB 缓冲区。
u32 USBD_EPTReadOUTData(USBD_EPTn_Enum EPTn, u32 *pTo, u32 len);	从端点缓冲区读 OUT 数据，清除 NAK 位。
u32 USBD_EPTReadMemory(USBD_EPTn_Enum EPTn, u32 *pTo, u32 len);	从端点缓冲区读 OUT 数据。

API 宏

USB 核心使用 API 宏作为与 USB 外设驱动层连接的接口，以保持它们之间的信息全透明。下表显示了 USB 核心使用的所有 API 宏。更详细的信息请参考“ht32fxxx_usb.h”文件。任何与 USB 核心层一起使用的外设驱动器，应在头文件中正确地定义特定的 API 宏。与 USB 外设驱动器一样，API 宏可以分为两种，全局和端点。所有的全局 API 宏都有“API_USB”前缀和一个便于理解的函数名。所有的端点 API 宏都有“API_USB_EPTn”前缀和一个便于理解的函数名。

表 6 USB 核心和 USB 外设驱动层之间的 API 宏

API 宏名称	描述
全局 API 宏	
API_USB_INIT(driver)	USB 外设初始化 API 宏通过 USBDCore_Init() 函数调用。
API_USB_DEINIT()	USB 外设取消初始化函数。当复位事件发生在 USB 总线，USBDCore_Reset() 函数将调用这个 API 宏。
API_USB_POWER_OFF()	此 API 宏用于关闭 USB 设备的电源。
API_USB_POWER_ON()	此 API 宏用于打开 USB 设备的电源。
API_USB_REMOTE_WAKEUP()	此 API 宏用于向 USB 主机发送恢复请求进行远程唤醒。
API_USB_READ_SETUP(buffer)	从 USB 缓冲区读取 SETUP 数据。
API_USB_SET_ADDR(addr)	设置 USB 设备地址。
API_USB_GET_CTRL_IN_LEN()	获取控制 IN 数据长度。
API_USB_ENABLE_INT(flag)	使能全局中断。
API_USB_GET_INT()	获取全局中断标志。
API_USB_GET_EPT_NUM(flag)	获取发生中断的有效 USB 端点编号。
API_USB_IS_RESET_INT(flag)	决定是否产生复位中断。
API_USB_CLR_RESET_INT()	清除复位中断。
API_USB_IS_SOF_INT(flag)	决定是否产生帧起始中断。
API_USB_CLR_SOF_INT()	清除帧起始中断。
API_USB_IS_RESUME_INT(flag)	决定是否产生恢复中断。
API_USB_CLR_RESUME_INT()	清除恢复中断。
API_USB_IS_SUSPEND_INT(flag)	决定是否产生暂停中断。
API_USB_CLR_SUSPEND_INT()	清除暂停中断。
API_USB_IS_EPTn_INT(flag, EPTn)	决定是否产生端点中断。
API_USB_CLR_EPTn_INT(EPTn)	清除端点中断。
端点 API 宏	
API_USB_EPTn_INIT(EPTn, driver)	端点初始化 API 宏通过 Set Configuration 标准命令调用。
API_USB_EPTn_RESET(EPTn)	端点重置到默认状态。
API_USB_EPTn_SEND_STALL(EPTn)	发送 STALL 到特定端点。

API 宏名称	描述
API_USB_EPTn_GET_INT(EPTn)	获取端点中断标志。
API_USB_EPTn_IS_IN_INT(flag)	决定是否产生端点 IN 中断。
API_USB_EPTn_CLR_IN_INT(EPTn)	清除端点 IN 中断标志。
API_USB_EPTn_IS_OUT_INT(flag)	决定是否产生端点 OUT 中断。
API_USB_EPTn_CLR_OUT_INT(EPTn)	清除端点 OUT 中断标志。
API_USB_EPTn_IS_INT(flag)	决定是否产生端点 IN 或 OUT 中断。
API_USB_EPTn_CLR_INT(EPTn)	清除端点 IN 或 OUT 中断标志。
API_USB_EPTn_GET_HALT(EPTn)	获取端点暂停状态。
API_USB_EPTn_SET_HALT(EPTn)	设置端点暂停状态。
API_USB_EPTn_CLR_HALT(EPTn)	清除端点暂停状态。
API_USB_EPTn_WAIT_STALL_SENT(EPTn)	等待 STALL 发送。
API_USB_EPTn_CLR_DTG(EPTn)	清除端点数据触发位。
API_USB_EPTn_GET_BUFFLEN(EPTn)	获取端点缓冲区长度。
API_USB_EPTn_GET_CNT(EPTn, type)	获取端点传输计数。
API_USB_EPTn_WRITE_IN(EPTn, from, len)	写 IN 数据到 USB 缓冲区。
API_USB_EPTn_READ_OUT(EPTn, to, len)	从端点缓冲区读 OUT 数据，清除 NAK 位。
API_USB_EPTn_READ_MEM(EPTn, to, len)	从端点缓冲区读 OUT 数据。

USB核心

USB 核心由“ht32_usbd_core.h”和“ht32_usbd_core.c”两个文件组成。所有的 USB 核心功能，如总线事件处理程序、标准请求、枚举和控制 IN/OUT，都可以在这一层实现。以下几节描述了变量结构、USB 核心操作以及 USB 核心 API。

表 7 USB 核心模块

模块	描述
ht32_usbd_core (.h, .c)	USB 核心模块，包括 USB 总线事件处理程序、标准请求、枚举和控制 IN/OUT 功能。

变量结构

对于 USB 核心操作，USB 核心层提供了一个名为“USBDCore_TypeDef”的结构，为 USB 核心函数提供必要的变量。用户应声明这个结构作为全局变量，并传送到“USBDCore_Init()”函数的存储地址。“USBDCore_TypeDef”结构也为应用层函数提供必要的变量，如“USBDDesc_Init()”和“USBDClass_Init()”函数。详细的使用信息稍后将在应用层讨论。下表显示了“USBDCore_TypeDef”结构的所有成员及其描述。更详细的信息请参考“ht32_usbd_core.c/h”文件。

表 8 USB 核心的变量结构

名称	类型		描述
USBCore.	Device.	Request.	bmRequestType: 请求特性。
(USBDCore_Type	(USBDCore_De	(USBDCore_Request	bRequest: 特定请求。

名称	类型		描述
Def)	vice_TypeDef)	_TypeDef)	wValueL: 请求参数低字节。
			wValueH: 请求参数高字节。
			wIndex: 请求参数。
			wLength: 数据传输阶段的字节数。
		Desc. (USBDCore_Desc_TypeDef)	pDeviceDesc: 设备描述符的数据指针。
			pConfnDesc: 配置描述符的数据指针。
			pStringDesc[n]: 字符串描述符的数据指针。
		Transfer. (USBDCore_Transfer_TypeDef)	uBuffer[2]: 临时缓冲区。
			pData: 控制 IN/OUT 数据的指针。
			sByteLength: 控制 IN/OUT 传输的总长度。
			Action: STALL、控制 IN 或 OUT。 (USBDCore_Action_Enum)
			CallBack_OUT.func/uPara: 控制 OUT 的回调函数指针和参数。
	Info. (USBDCore_Info_TypeDef)		uCurrentConfiguration: 用于设置/获取配置请求。
			uCurrentInterface: 用于设置/获取接口请求。
			CurrentStatus: 设备的状态。(USBDCore_Status_Enum)
			LastStatus: 设备暂停前的状态。(USBDCore_Status_Enum)
			CurrentFeature: 用于设置/清除特性, 并获取状态请求。 (USBDCore_Feature_TypeDef)
			uIsDiscardClearFeature: 放弃清除大容量存储的特性请求。
	Class. (USBDCore_Class_TypeDef)		CallBack_MainRoutine.func/uPara: 类主程序的回调函数/参数。
			CallBack_StartOfFrame: 类 SOF 的回调函数。
			CallBack_ClassGetDescriptor: 类获取描述符的回调函数。
			CallBack_ClassSetInterface(pDev): 类设置接口的回调函数。
			CallBack_ClassRequest(pDev): 类请求回调函数。
			CallBack_EPTn[MAX_EP_NUM](EPTn): 端点 n 回调函数。
	pDriver (u32)		USB 设备驱动器的初始化结构。
	Power. (USBDCore_Power_TypeDef)		CallBack_Suspend.func/uPara: 暂停时系统低功耗函数/参数。

USB 核心操作

当特定的 USB 总线事件发生后，USB 设备 IP 发出中断信号，通知 CPU 发生了事件，需要处理。中断控制器和 CPU 完成所有必要的操作后，CPU 开始执行中断服务程序（ISR），来处理 USB 总线事件。USB 核心层提供了 ISR 函数“USBDCore_IRQHandler()”来完成上述操作。

“USBDCore_IRQHandler()”通过 API 宏（外设驱动层）检查 USB 设备 IP 的状态寄存器，以确定 USB 总线事件，然后调用相应的函数来处理它。下表给出了所有 USB 总线事件及其相应函数。详细的操作请参考“ht32_usbd_core.c”文件中的“USBDCore_IRQHandler()”函数。

表 9 USB 总线事件和 USB 核心层的相应函数

总线事件	相应函数	描述
复位	_USBDCore_Reset(pCore);	当 USB 总线复位事件发生，USB 核心将重置所有状态到默认值，打开 USB IP 电源和最初的控制端点，并使能 USB 中断。上述操作完成后，控制端点将通过 USB 地址 0 准备接收/发送数据。全局状态变量和 USB 设备的 IP 控制寄存器也被重置。
恢复	_USBDCore_Resume(pCore);	当 USB 总线恢复事件发生，USB 核心将打开 IP 电源，恢复当前的状态变量（恢复到进入暂停模式之前的值）。
暂停	_USBDCore_Suspend(pCore);	当 USB 总线暂停事件发生，USB 核心将改变当前的状态变量为“暂停”。它使 USB 核心执行应用层的暂停回调函数，让芯片进入低功耗模式。
控制端点 SETUP 包准备就绪	_USBDCore_Setup(pCore);	当 SETUP 数据已准备就绪，USB 核心使用此函数来处理 SETUP 数据。此函数需要考虑 USB 标准、类和供应商的要求。
控制端点 IN 数据传输	_USBDCore_ControlIN(pCore);	当最后一笔控制 IN 数据传输完毕后，USB 核心使用此函数来处理下一笔 IN 数据。此函数必要时也需要考虑零长度包（ZLP）。
控制端点 OUT 数据准备就绪	_USBDCore_ControlOUT(pCore);	当接收到控制 OUT 数据，USB 核心使用此函数来处理 OUT 数据。
端点 n IN 数据传输 端点 n OUT 数据准备就绪	pCore->Class.CallBack_EPTnEPTn;	回调函数需考虑端点 IN 或 OUT 的数据传输，通过“USBDClass_Init()”函数设置。

USB 核心层提供了“回调”能够处理一些在已知的情况下需要涉及到应用层的总线事件。例如，帧起始、获取类描述符、设置接口或类请求，可能需要除 USB 核心层外的操作。在类初始化阶段，用户应指定回调函数给 USB 核心层。然而，在某些特定情况下，用户可能需要修改 USB 核心层，以适应类应用的要求。但不得经常发生。

USB 核心 API

USB 核心层为应用层提供以下 API 实现类应用功能。对于统一的用户界面，USB 核心 API 函数并没有真的存在，它只是一个重新定义 API 名称的 API 宏（USB 外设驱动层）。请注意，应用层应避免直接使用除 USB 核心 API 外的其它任何函数。因此，它可以保持每一层的独立性，提高模块更换的可能性。

表 10 USB 核心 API

API 名称	描述
初始化和系统 API	
void USBDCore_Init(USBDCore_TypeDef *pCore);	USB 核心初始化。
void USBDCore_IRQHandler(USBDCore_TypeDef *pCore);	USB 核心的中断服务程序。
void USBDCore_MainRoutine(USBDCore_TypeDef *pCore);	USB 核心的应用主程序。
u32 USBDCore_IsSuspend(USBDCore_TypeDef *pCore);	返回暂停状态。
u32 USBDCore_GetRemoteWakeUpFeature(USBDCore_TypeDef *pCore);	返回远程唤醒状态，通过 SET FEATURE 标准命令设置。
void USBDCore_TriggerRemoteWakeup(void);	打开 USB 电源，远程唤醒主机。
端点 API	
void USBDCore_EPTReset(USBD_EPTn_Enum EPTn);	端点重置到默认状态。
u32 USBDCore_EPTGetBufferLen(USBD_EPTn_Enum EPTn);	获取端点缓冲区长度。
u32 USBDCore_EPTGetTransferCount(USBD_EPTn_Enum EPTn, USBD_TCR_Enum type);	获取端点传输计数。
void USBDCore_EPTSetSTALL(USBD_EPTn_Enum EPTn);	设置端点暂停状态。
Void USBDCore_EPTWaitSTALLSent(USBD_EPTn_Enum EPTn);	等待 STALL 发送。
Void USBDCore_EPTClearDataToggle(USBD_EPTn_Enum EPTn);	清除端点暂停状态。
端点数据 API	
u32 USBDCore_EPTWriteINData(USBD_EPTn_Enum EPTn, u32 *pFrom, u32 len);	写 IN 数据到 USB 缓冲区。
u32 USBDCore_EPTReadOUTData(USBD_EPTn_Enum EPTn, u32 *pTo, u32 len);	从端点缓冲区读 OUT 数据，清除 NAK 位。
u32 USBDCore_EPTReadMemory(USBD_EPTn_Enum EPTn, u32 *pTo, u32 len);	从端点缓冲区读 OUT 数据。

应用层

应用层包括用于配置、描述符、类请求和端点数据处理的几个文件。USB 类的主要功能设在这一层。用户可以在应用层修改文件，以适应应用需求。下表显示了应用层的模块和文件。每个模块的详细信息将在本节稍后讨论。本节还提供如何配置以及最初的 USB 核心和描述符的详细信息。

表 11 应用层模块

模块	描述
ht32fxxxx_usbdconf.h	USB 设备的配置文件, 包括与 USB 设备 IP 相关的中断和端点设置。
ht32_usbd_descriptor (.c, .h)	USB 设备的描述符, 包括设备、配置和字符串。此模块提供了初始化函数用于设置描述符的数据指针, 为 USB 核心获取描述符请求。
ht32f_usbd_class (.c, .h)	类相关功能, 包括初始化、主程序、类标准请求、类请求和端点数据处理。

配置文件 (ht32fxxxx_usbdconf.h)

配置文件用于配置 HT32 USB 外设, 包括全局中断使能位和端点设置。下图为 KEIL uVision IDE 的配置向导窗口。用户还可以通过直接编辑 “ht32fxxxx_usbdconf.h” 文件改变设置。注意, 带 “(Default)” 标志的项目在一般情况下不得被禁用。

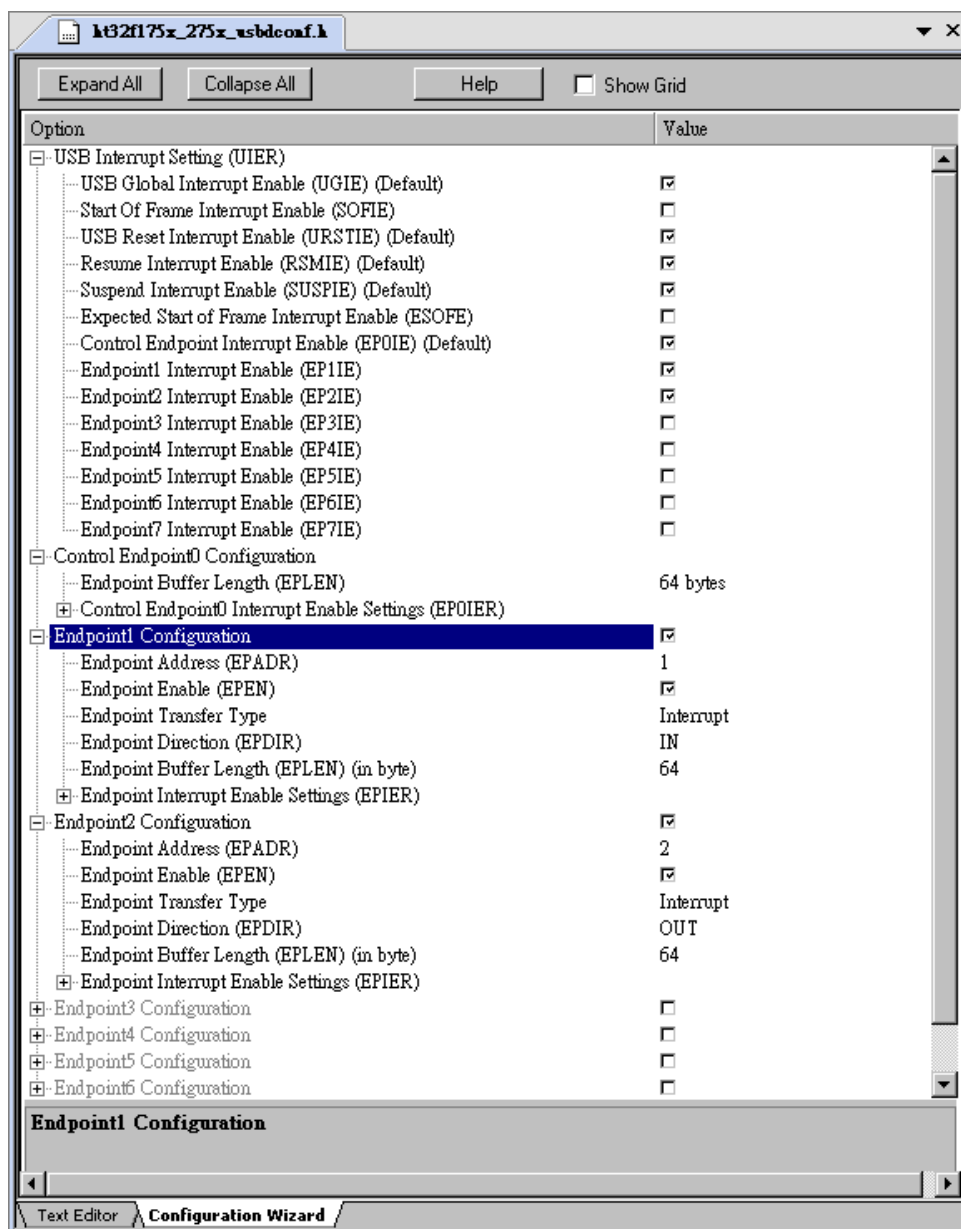


图 3 Keil uVision IDE 的配置向导

下列步骤显示了配置流程和注意事项。

- 对于 USB 中断设置 (UIER)，应确保带“(Default)”标志的项目使能。根据类/应用的端点使用，使能端点 n 中断 (EPnIE)。
- 根据 USB 缓冲存储器的使用情况，配置控制端点 0 的缓冲区长度为 8、16、32 或 64 字节。
- 确保端点 0 的 ODRXIE、IDTXIE 和 SDRXIE 中断使能位开启。USB 核心层依靠这些总线事件来处理 USB 控制传输。
- 根据类/应用的端点使用来配置端点 n 设置。下表显示了应要注意的设置。

表 12 端点设置

设置	描述
端点地址	一般情况下不需要修改。
端点使能	端点使用前应打开。

设置	描述
端点传输类型	端点 1 ~ 3: 批量或中断 端点 4 ~ 7: 等时, 批量或中断。
端点方向	IN 或 OUT。
端点缓冲区长度	中断: 1 ~ 64 字节 批量: 8, 16, 32 或 64 字节 等时: 1 ~ 1023 字节。(注意: 因为 USB 缓冲区至少保留 8 + 8 字节用于 SETUP 数据和端点 0, 所以等时端点的实际最大缓冲区长度为 1008 字节)
端点中断使能	对于 IN 端点, 固件通常使用 IN 数据包传送中断使能 (IDTXIE) 来处理数据传输程序。对于 OUT 端点, 固件通常使用 OUT 数据包接收中断使能 (ODRXIE) 来处理数据传输程序。其它中断可用于调试或其它用途。更多详细信息, 请参考 HT32Fxxxx 用户手册。

描述符设置

设备、配置、接口、端点和字符串描述符都包含在应用层的“ht32_usbd_descriptor.c”文件中。用户可根据类/应用的要求修改它们。下图显示了设备 KEIL uVision IDE 的配置向导和配置描述符。对于其它 IDE 或描述符, 用户可以通过直接编辑“ht32_usbd_descriptor.c”文件修改描述符。

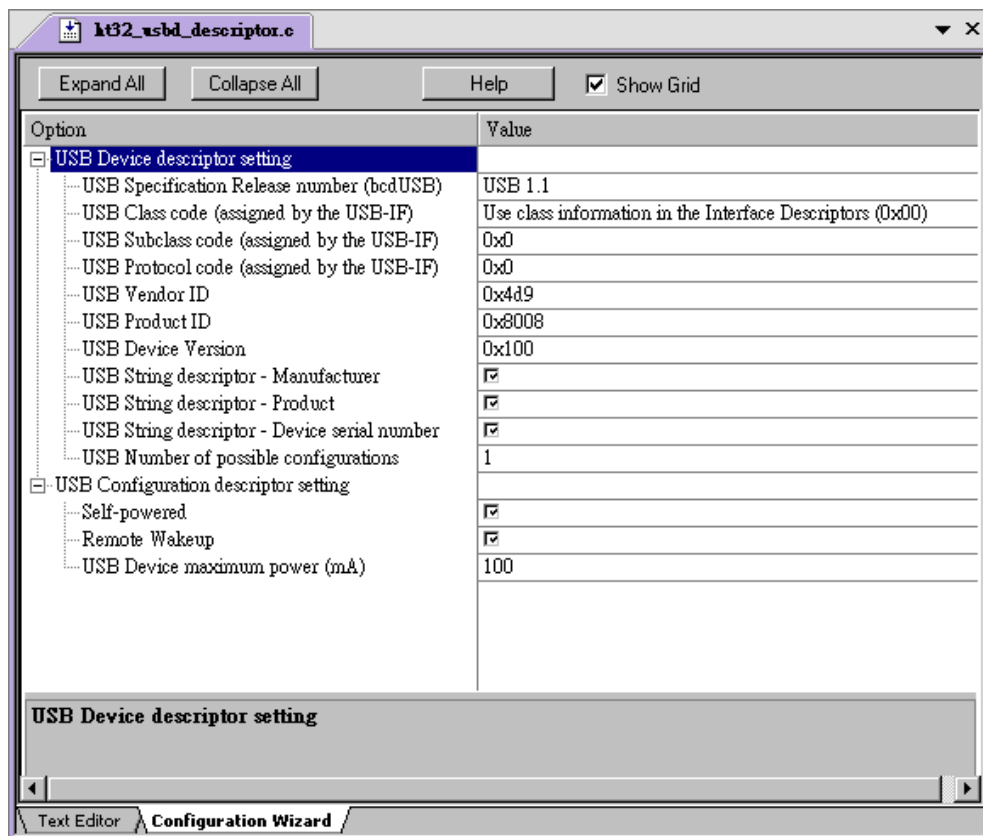


图 4 设备 Keil uVision IDE 的配置向导和配置描述符

初始化流程

下面的源代码，给出了一个如何初始化 USB 描述符、类和核心的例子。初始化流程完成后，应用程序应调用“USBDCore_MainRoutine(&gUSBCore)”函数定期处理暂停回调函数和类主程序。

```
__ALIGN4 USBDCore_TypeDef gUSBCore;          /* USB 核心存储器,必须 4 字节对齐 */
USB_Driver_TypeDef gUSBDriver;               /* USB 驱动存储器 */
void USB_Configuration(void)
{
    gUSBCore.pDriver = (u32 *)&gUSBDriver;    /* 初始化 USB 驱动的存储指针 */
    gUSBCore.Power.CallBack_Suspend.func = Suspend; /* 安装暂停回调函数到 USB 核心 */

    USBDDesc_Init(&gUSBCore.Device.Desc);      /* 初始化描述符的存储指针 */
    USBDClass_Init(&gUSBCore.Class);           /* 初始化 USB 类层 */
    USBDCore_Init(&gUSBCore);                  /* 初始化 USB 核心层 */

    NVIC_EnableIRQ(USB_IRQn);                 /* 使能 USB 设备中断 */
}
int main(void)
{
    ...
    USB_Configuration();                      /* USB 相关配置 */
                                              /* 连接 USB, 并等待枚举完成 */
    HT32F_DVB_USBConnect();
    while(gUSBCore.Info.CurrentStatus != USB_STATE_CONFIGURED);
    ...
    while (1)
    {
        ...
        USBDCore_MainRoutine(&gUSBCore);      /* USB 核心主程序 */
        ...
    }
    ...
}
```

类实现

本节提供了类实现的详细信息，包括类请求、端点数据处理和类主程序。本节的目的是描述如何修改应用层，以适应 USB 类请求。

• 类的初始化

类的初始化安装相关回调函数/参数，包括：

- ◆ 类主程序
- ◆ 帧起始
- ◆ 标准获取描述符请求类
- ◆ 标准设置接口请求
- ◆ 类请求
- ◆ 端点处理程序

初始化函数“USBDClass_Init”可以在每个类范例代码的“ht32_usbd_class.c”文件中找到。

用户可以按照以下范例来安装其类/应用的回调函数。

```

/*****
 * @brief USB Class initialization.
 * @param pClass: pointer of USBDCore_Class_TypeDef
 * @retval None
 *****/
void USBDClass_Init(USBDCore_Class_TypeDef *pClass)
{
    pClass->CallBack_MainRoutine.func = USBDClass_MainRoutine;
    //pClass->CallBack_MainRoutine.uPara = (u32)NULL;

    //pClass->CallBack_StartOfFrame.func = USBDClass_StartOfFrame;
    //pClass->CallBack_StartOfFrame.uPara = (u32)NULL;

    pClass->CallBack_ClassGetDescriptor = USBDClass_Standard_GetDescriptor;
    //pClass->CallBack_ClassSetInterface = USBDClass_Standard_SetInterface;

    pClass->CallBack_ClassRequest = USBDClass_Request;
    pClass->CallBack_EPTn[1] = USBDClass_Endpoint1;
    pClass->CallBack_EPTn[2] = USBDClass_Endpoint2;
    //pClass->CallBack_EPTn[3] = USBDClass_Endpoint3;
    //pClass->CallBack_EPTn[4] = USBDClass_Endpoint4;
    //pClass->CallBack_EPTn[5] = USBDClass_Endpoint5;
    //pClass->CallBack_EPTn[6] = USBDClass_Endpoint6;
    //pClass->CallBack_EPTn[7] = USBDClass_Endpoint7;

    return;
}

```

图 5 类的初始化范例

• 类请求

当 USB 核心收到 SETUP 传输和解码类请求，USB 核心将执行类请求回调函数（CallBack_ClassRequest）。USB 核心类请求通过“USBDCore_Device_TypeDef”结构点作为回调函数的参数。更多信息请参考“表 8 USB 核心的变量结构”。

“ht32_usbd_class.c”文件中的“USBDClass_Request()”函数，给出了一个如何处理类请求的例子。用户可以根据其要求实现自己的类请求函数。类请求函数应设置作为“USBDClass_Init”函数的回调函数，如下所示。

```

static void USBDClass_Request(USBDCore_Device_TypeDef *pDev);
void USBDClass_Init(USBDCore_Class_TypeDef *pClass)
{
    ...
    pClass->CallBack_ClassRequest = USBDClass_Request;
    ...
}

```

• 端点处理程序

对于 USB 的特性，固件通常配置 USB 设备控制器在 IN 数据包发送或收到 OUT 数据包时，发出端点中断。当端点中断发生，USB 核心将检查中断标志，并执行相应的端点处理程序。端点处理程序负责端点数据处理。以下 USB 核心函数用于端点数据处理。

表 13 USB 核心数据处理 API

API 名称	描述
USBDCore_EPTWriteINData	写 IN 数据到 USB 缓冲区。
USBDCore_EPTRReadOUTData	从端点缓冲区读 OUT 数据，清除 NAK 位。
USBDCore_EPTRReadMemory	从端点缓冲区读 OUT 数据。

固件可处理端点处理程序的内部数据，或只设置一个软件标志位，以通知应用 IN/OUT 数据。注意单片机在中断模式下执行端点处理程序，将根据优先级阻止其它中断。如果 USB 数据

很大，我们应当防止在端点处理程序内部处理数据。用户应当设置端点处理程序作为“USBDClass_Init”函数的回调函数，如下所示。

```
static void USBDClass_Request(USBDCore_Device_TypeDef *pDev);
void USBDClass_Init(USBDCore_Class_TypeDef *pClass)
{
    ...
    pClass->CallBack_EPTn[1] = USBDClass_Endpoint1;
    pClass->CallBack_EPTn[2] = USBDClass_Endpoint2;
    //pClass->CallBack_EPTn[3] = USBDClass_Endpoint3;
    //pClass->CallBack_EPTn[4] = USBDClass_Endpoint4;
    //pClass->CallBack_EPTn[5] = USBDClass_Endpoint5;
    //pClass->CallBack_EPTn[6] = USBDClass_Endpoint6;
    //pClass->CallBack_EPTn[7] = USBDClass_Endpoint7;
    ...
}
```

• 类主程序

如果固件在 USB ISR 期间花了太多时间在端点数据处理过程上，那么一些优先级较低的中断将不能被 MCU 响应。它可能不适合一些需要特定响应时间的外设。因此，我们建议在端点处理程序外部来处理这些低优先级的端点数据。“USBDClass_MainRoutine()”可以帮助用户实现。当端点中断发生时，用户可以将端点处理程序中的软件标志置高。USB ISR 返回后，检查标志位，处理“USBDClass_MainRoutine()”函数中的 IN 或 OUT 数据。同时，对于带 DMA 控制器的单片机，使用 DMA 来防止上述情况也是一个好办法。当 DMA 传输完成时，可以触发端点处理程序的 DMA，处理“USBDClass_MainRoutine()”函数中的数据。USB 核心将定期执行“USBDClass_MainRoutine()”函数。用户应当设置它作为“USBDClass_Init”函数的回调函数。回调函数保留“uPara”参数为将来使用。更多信息请参考“ht32_usbd_class.c”文件。

```
static void USBDClass_MainRoutine(u32 uPara)
void USBDClass_Init(USBDCore_Class_TypeDef *pClass)
{
    ...
    pClass->CallBack_MainRoutine.func = USBDClass_MainRoutine;
    //pClass->CallBack_MainRoutine.uPara = (u32)NULL;
    ...
}
```

使用HT32 USB设备的类范例代码

本章介绍类范例代码的基本用法，包括项目、Windows 驱动器和 USB 核心调试模式。

项目文件

类范例代码提供 KEIL MDK-ARM、IAR EWARM 和 CooCox CoIDE 的项目文件，包括：

- USB/CLASS_EXAMPLE/CoIDE_EXAMPLENAME/CoIDE_EXAMPLENAME.cob
- USB/CLASS_EXAMPLE/EWARM/Project.eww
- USB/CLASS_EXAMPLE/MDK_ARM/Project.uvproj
- USB/CLASS_EXAMPLE/SourceryG++Lite/Project.uvproj

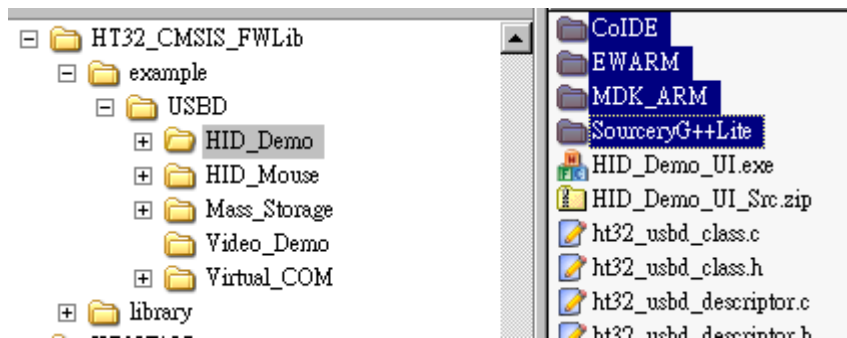


图 6 类范例代码的项目文件

可以打开每个类范例代码的项目文件的首选 IDE 在开发板上进行测试。请参考每个 IDE 快速入门指南的编译、下载和调试操作。

Windows的USB驱动

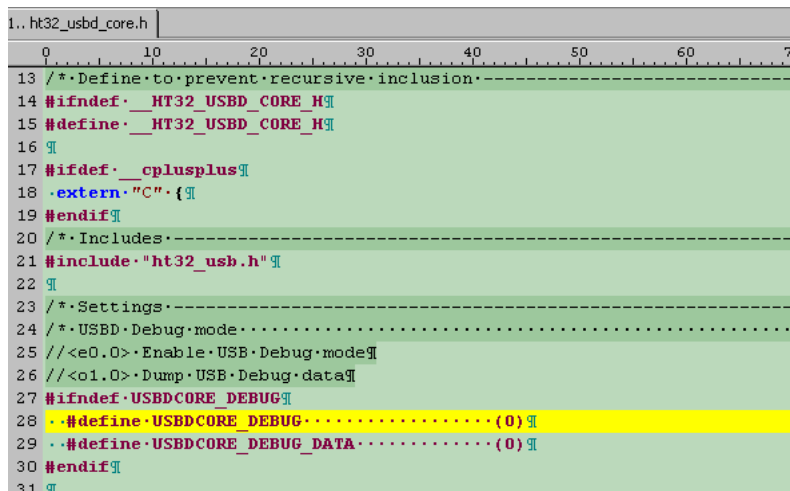
所有的类范例代码（除了虚拟 COM 端口范例）使用 Windows 内置驱动器连接到你的计算机。你不必为它们指定驱动器。虚拟 COM 端口范例使用由 Holtek 所提供的“HT32_VCP.inf”文件，告诉 Windows 应该加载哪个驱动器。当连接虚拟 COM 端口范例到 Windows 时，需要指定“HT32_VCP.inf”的路径。“HT32_VCP.inf”位于“{Firmware Library Path}\xxxx_CMSIS_FWLib\example\USB\Virtual_COM\”。

所有的类范例代码使用相同的 VID 和 PID，0x04D9 和 0x8008。当使用相同的 VID 和 PID 连接不同的类范例代码到同一台计算机上，Windows 可能无法正确加载驱动器。同样，如果修改描述符，但保持相同的 VID 和 PID，上述情况也可能发生。Windows 设备管理器显示 USB 设备“故障”。下面步骤显示了如何解决 Windows XP 下的上述问题。在其他版本的 Windows 或其他操作系统，你可能需要寻找相同的功能。

- 按控制面板系统对话框中“硬件”选项卡的按钮，打开“设备管理器”。
- 在列表找出插入到计算机的 USB 设备（带“故障”图标）。
- 在设备上单击右键，选择“卸载”，然后从计算机上拔下并重新连接。
- Windows 自动重新安装正确的 USB 设备驱动器。

USB核心调试模式

用于调试目的，可以通过设置“USBDCORE_DEBUG”的定义值为1（在ht32_usbd_core.h），打开USB核心调试模式。根据固件库的重定向设置，USB核心使用printf函数通过UART或ITM接口显示调试信息。需要注意的是USB核心调试模式会降低性能，在一定条件下可能会导致USB出错。在最终应用上应关闭USB调试模式。我们强烈建议当USB调试消息被重定向到UART接口时，关闭“转存USB调试数据（USBDCORE_DEBUG_DATA）”选项。



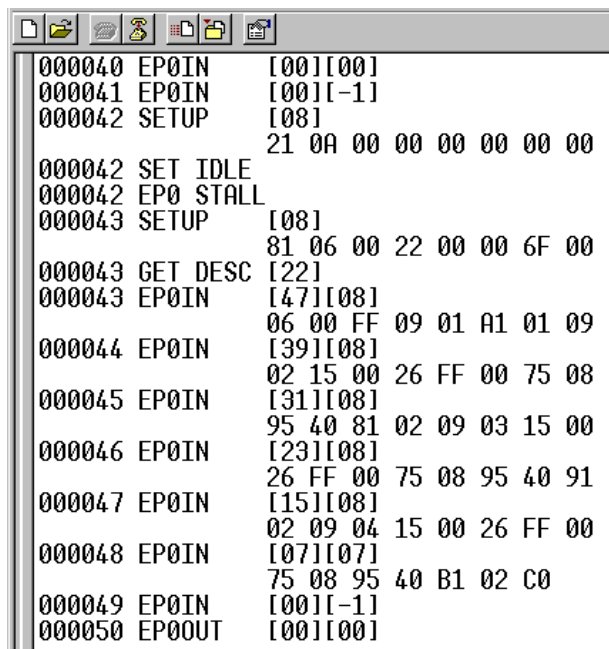
```

1.. ht32_usbd_core.h
13 /* Define to prevent recursive inclusion */
14 #ifndef __HT32_USBD_CORE_H__
15 #define __HT32_USBD_CORE_H__
16
17 #ifndef __cplusplus
18 .extern "C" {
19 #endif
20 /* Includes */
21 #include "ht32_usb.h"
22
23 /* Settings */
24 /* USBDCORE_Debug mode */
25 // <0> Enable USBDCORE_Debug mode
26 // <1> Dump USBDCORE_Debug data
27 #ifndef USBDCORE_DEBUG
28 ..#define USBDCORE_DEBUG ..... (0)
29 ..#define USBDCORE_DEBUG_DATA ..... (0)
30 #endif
31

```

图7 USB核心调试模式设置

图8显示了USB核心调试模式的输出范例。



```

000040 EP0IN [00][00]
000041 EP0IN [00][-1]
000042 SETUP [08]
21 0A 00 00 00 00 00 00
000042 SET IDLE
000042 EP0 STALL
000043 SETUP [08]
81 06 00 22 00 00 6F 00
000043 GET DESC [22]
000043 EP0IN [47][08]
06 00 FF 09 01 A1 01 09
000044 EP0IN [39][08]
02 15 00 26 FF 00 75 08
000045 EP0IN [31][08]
95 40 81 02 09 03 15 00
000046 EP0IN [23][08]
26 FF 00 75 08 95 40 91
000047 EP0IN [15][08]
02 09 04 15 00 26 FF 00
000048 EP0IN [07][07]
75 08 95 40 B1 02 C0
000049 EP0IN [00][-1]
000050 EP0OUT [00][00]

```

图8 USB核心调试模式的输出范例

人机接口设备 (HID)

简介

人机接口设备 (HID) 是专为人类控制计算机主机设计的。HID 使用报告的概念, 通过控制或中断管道交换数据。由于大多数操作系统支持内置 HID 驱动器, 我们可以使用 HID 类的方便方式与计算机交换数据, 而无需执行任何操作系统的驱动程序。USB 设备 HID 的范例代码显示了如何使用 HT32 USB 设备库、枚举、端点中断传输和 HID 获取/设置报告。这个例子只是说明如何回送 HID 输出报告给输入报告以及设置特性报告回送给获取特性报告。

使用HID范例代码

HID 的范例代码在 HT32 系列开发板实现 USB HID 类。当范例代码下载到开发板, USB 连接到 PC, 你可以在设备管理器中找到一个新的 HID 设备 (VID = 0x04D9, PID = 0x8008), 如下图所示。

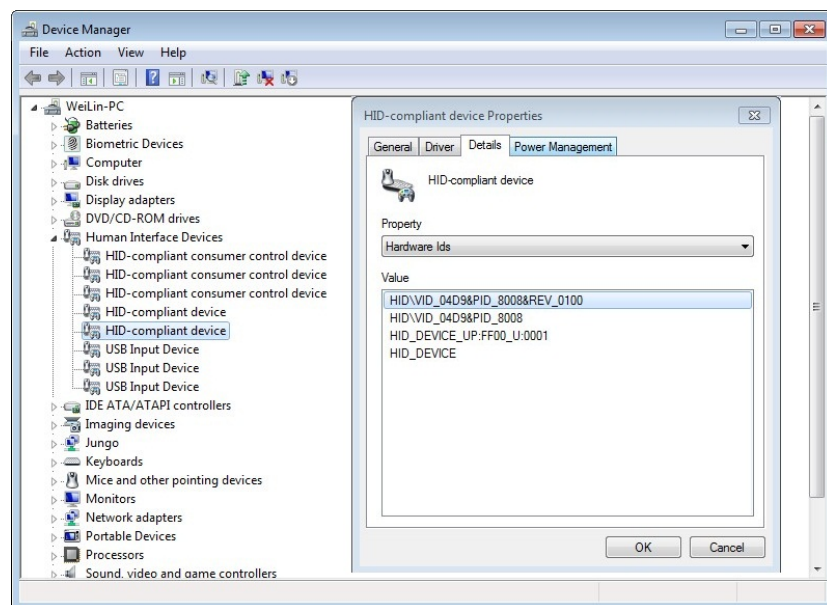


图 9 设备管理器的 HID 设备

HID 范例代码还提供基于 PC 的演示 UI “HID_Demo_UI.exe”, 显示了 HID 设备如何与 PC 交换数据。演示 UI 基于 Microsoft Virtual C++ 2005。HID 演示 UI 的源代码位于 HID 范例代码路径中的 “HID_Demo_UI_Src.zip” 文件中。用户可以根据 HID 演示 UI 范例代码开发定制基于 PC 的应用。

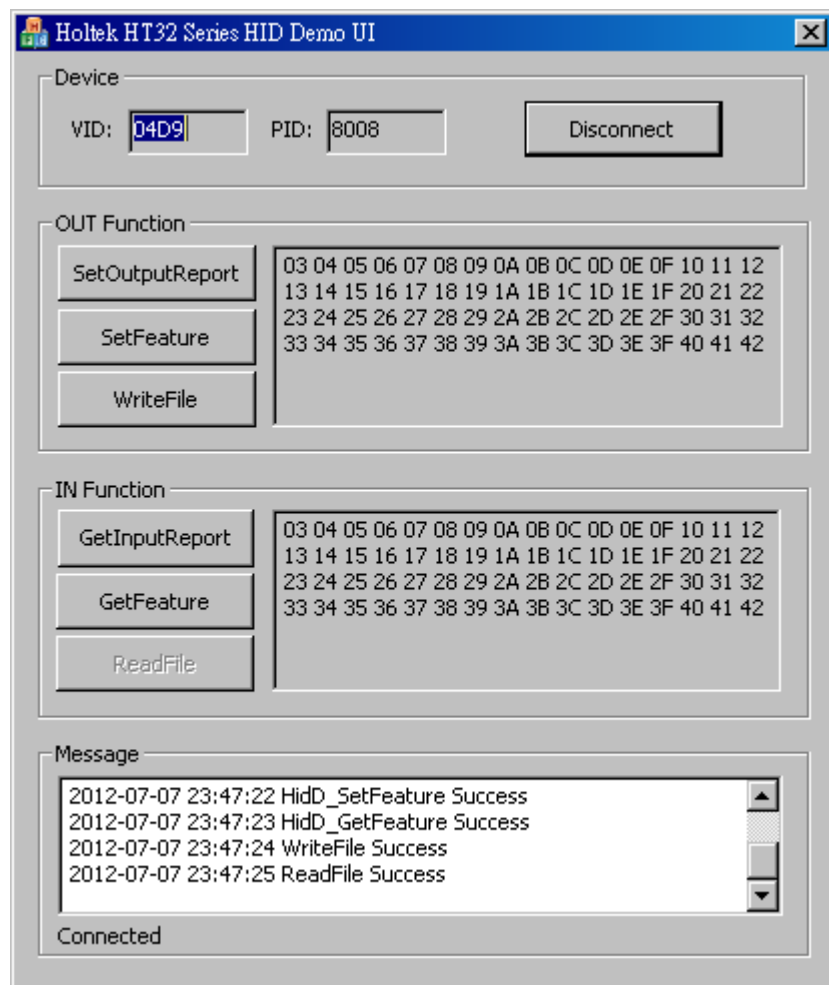


图 10 基于 PC 的 HID 范例代码的演示 UI

HID类的实现

本节介绍 HID 类的实现，包括 HID 报告描述符、端点配置、类请求和数据传输。

HID 报告描述符

HID 报告描述符，“USB_HID_ReportDesc[]”数组位于“ht32_usnd_class.c”文件中，定义了 HID 报告的格式和目的。更多信息，请参考 HID 规格书。USB 实施者论坛提供了一个名为“HID Descriptor Tool”的工具可自动生成 HID 报告描述符。你可以使用这个工具生成 HID 报告描述符，并保存为头文件（*.H）。头文件中报告描述符的数组数据可以直接复制和粘贴到“USB_HID_ReportDesc[]”数组。请注意，HID 报告描述符的长度定义（DESC_LEN_RPOT）位于“ht32_usbd_class.h”文件中。当 HID 报告描述符的长度改变时，必须修改。

```

36
37 /* HID related definition
38 #define DESC_LEN_HID                ((u32) (9))
39 #define DESC_LEN_RPOT                ((u16) (47))
40
41 #define DESC_TYPE_21_HID              (0x21)
42 #define DESC_TYPE_22_RPOT            (0x22)
43 #define DESC_TYPE_23_PHY              (0x23)

```

图 11 HID 报告描述符的长度定义

端点配置

HID 的范例代码使用三个端点用于控制和报告传输。下表显示了端点的数量、方向和传输类型。端点的详细设置，可以在“ht32xxx_usbdconf.h”文件中找到。

表 14 端点的 HID 范例代码

端点号	传输类型
端点 0	控制 IN 或 OUT
端点 1	IN 中断
端点 2	OUT 中断

- 端点 0（控制）：用于 USB 标准/类请求。使用此端点进行设置报告和获取报告的数据传输。
- 端点 1（IN 中断）：用于 HID 输入报告的数据传输。
- 端点 2（OUT 中断）：用于 HID 输出报告的数据传输。

类请求

USB HID 设备定义了六个类专用请求。这些类请求可以在 HID 范例代码的“ht32_usbd_class.c”文件中执行。不是所有的命令都是必需的。下表显示了所有已定义的 HID 类请求和在此范例代码中其执行情况。

表 15 HID 类请求

类请求	描述	请求?	执行?
获取报告	主机通过控制管道接收来自设备的输入或特性报告。	Y	Y
获取空闲	主机读取当前闲置率。	N	N
获取协议	主机读取有效协议（启动或报告）。	N(*)	N
设置报告	主机通过控制管道发送输出或特性报告给设备。	N	Y
设置空闲	主机设置设备的闲置率以节省带宽。	N	N
设置协议	主机切换设备的启动或报告协议。	N(*)	N

*注：启动设备所需

数据传输

下表显示了 HID 演示范例代码的数据传输流程。更多详细信息，请参考端点处理程序和“ht32_usbd_class.c”文件的类主程序。

表 16 演示范例代码的数据传输流程

主机	设备	动作
设置输出报告	端点 0（控制）	保存到 gOutputReportBuffer，并复制到 gInputReportBuffer
输出报告（读文件）	端点 2（OUT）	
获取输入报告	端点 0（控制）	从 gInputReportBuffer 中传输数据
输入报告（写文件）	端点 1（IN）	
设置特性报告	端点 0（控制）	保存到 gFeatureReportBuffer
获取特性报告	端点 0（控制）	从 gFeatureReportBuffer 中传输数据

大容量存储

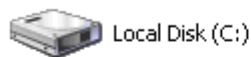
简介

USB 设备大容量存储的范例代码显示了如何使用 HT32 USB 设备库、枚举和端点批量传输。此范例代码使用仅批量传输（BOT）协议与 PC 主机通信。媒体访问、SCSI（小型计算机系统接口）块命令、RAM 磁盘和 SD 卡相关的功能也可以在此范例代码中实现。此范例代码能够支持 RAM 磁盘和 SD 卡。RAM 磁盘的范例可以帮助用户了解 USB 大容量存储、BOT 方式和 SCSI 命令的架构。RAM 磁盘功能也是一个很好的起点，以创建自定义的数据交换应用。客户的数据交换可以利用大容量存储的好处，如快速批量传输和大多数操作系统无需驱动。

使用大容量存储范例代码

请参考第 3 章，建立和下载大容量存储的范例代码到 HT32 系列开发板，插入 SD 卡，连接 USB 到计算机主机。当 USB 枚举完成后，可以通过 Windows 文件资源管理访问 RAM 磁盘和 SD 卡。数据可以通过文件资源管理从 SD 卡读取或写入。可以用于通过 PC 更新 SD 卡内容。

Hard Disk Drives



Devices with Removable Storage



图 12 Windows 中的 RAM 磁盘和 SD 卡的大容量存储范例代码

RAM 磁盘有一个“README.TXT”文件，其中包含范例文本数据。你可以简单地双击打开这个文件。它显示了如何通过 RAM 磁盘功能进行 PC 主机和 USB 设备之间的数据交换。

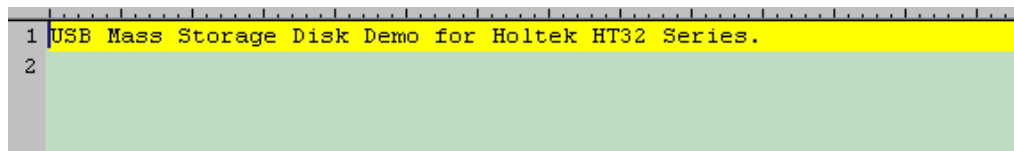


图 13 “README.TXT”的范例文本数据

大容量存储类的实现

大容量存储范例实现了仅批量传输（BOT）协议、SCSI 块命令（SBC-3 和 SPC-4）、RAM 磁盘和 SD 卡的相关功能。本节说明了上述功能的基本信息。大容量存储的类范例代码可以分为三层，BOT 协议、SCSI 块命令和媒体访问。下表显示了大容量存储的范例代码模块。

表 17 大容量存储的范例代码模块

模块	描述
ht32f_usbd_class (.c, .h)	作为 USB 核心和 BOT 协议的中间层。它负责初始化、数据传输和类请求。
usb_bulk_only_transport (.c, .h)	BOT 协议命令处理程序和功能。
usb_scsi_block_command (.c, .h)	SCSI 块命令处理程序和功能。
usb_scsi.h	SCSI 协议的相关定义、结构和常数。
MEDIA_disk (.c, .h)	媒体访问的相关功能。

范例代码支持透明的 SCSI 命令集 (0x06, 由配置描述符的 “bInterfaceSubClass” 子类代码设置) 和 USB 大容量存储类仅批量传输 (0x50 BBB 协议, 由配置描述符的 “bInterfaceProtocol” 设置)。以下各节提供了 BOT 和 SCSI 命令的简要说明。更多信息请参考各个规格书。

仅批量传输 (BOT)

仅批量传输协议使用批量 IN 和 OUT 端点来传输命令块包 (CBW)、命令状态包 (CSW) 和 IN/OUT 数据。BOT 协议的传输不需要控制和中断管道。大容量存储 BOT 的执行分为三个阶段, 命令阶段 (CBW), 可选数据阶段 (IN 或 OUT) 和状态阶段 (CSW)。BOT 流程如下。

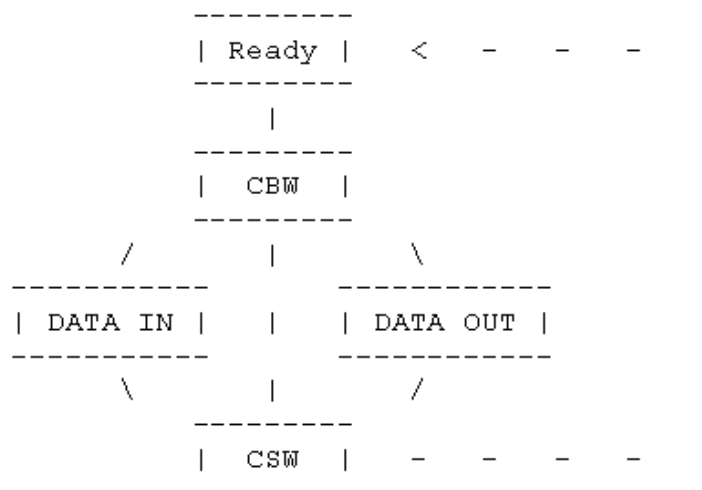


图 14 大容量存储 BOT 协议流程

以下两图显示了 CBW 和 CSW 的格式。更多详细信息, 请参考仅批量传输规格书。

bit	7	6	5	4	3	2	1	0
Byte								
0-3	dCBWSignature							
4-7	dCBWTag							
8-11 (08h-0Bh)	dCBWDataTransferLength							
12 (0Ch)	bmCBWFlags							
13 (0Dh)	Reserved (0)				bCBWLUN			
14 (0Eh)	Reserved (0)			bCBWCBLength				
15-30 (0Fh-1Eh)	CBWCB							

图 15 命令块包的格式

bit	7	6	5	4	3	2	1	0
Byte								
0-3	<i>dCSWSignature</i>							
4-7	<i>dCSWTag</i>							
8-11 (8-Bh)	<i>dCSWDataResidue</i>							
12 (Ch)	<i>bCSWStatus</i>							

图 16 命令状态包的格式

BOT CBW 包括设备要执行命令块的 CBWCB 领域。在这个范例代码中，CBWCB 执行 SCSI SBC-3 和 SPC-4 命令将被传递到 SCSI 层，在“usb_bulk_only_transport.c”文件中执行所有必需的函数。下表显示了函数及其说明。

表 18 BOT 层的 API

函数描述
void BOT_Init(void); BOT 初始化函数。包括 CSW 领域、SCSI 层和媒体。
void BOT_Reset(void); BOT 复位函数，用于仅批量传输大容量存储复位类请求。
uc8 *BOT_GetMaxLUNAddress(void); 获取最大逻辑单元号（LUN）。
void BOT_OUTProcess(void); 此函数用于处理 USB OUT 数据。此函数应在端点处理程序或类主程序中执行。
void BOT_INProcess(void); 此函数用于处理 USB IN 数据。此函数应在端点处理程序或类主程序中执行。

函数描述

u32 BOT_CheckCBWParameter(BOT_Dir_Enum DeviceDirection, u32 uDeviceLength);

此函数用于检查 CBW 参数是否有效 (Cass1 ~ Cass13)。

void BOT_WriteINData(u8 *pBuffer, u32 uLen, BOT_State_Enum status);

写 BOT IN 数据和更新 BOT 状态/CSW。

void BOT_ReadOUTData(u8 *pBuffer, u32 uLen);

读 BOT OUT 数据和更新 CSW。

void BOT_SendCSW(BOT_CSW_STATUS_Enum status);

此函数用于向主机发送 CSW。

void BOT_ErrorHandler(BOT_STALL_Enum StallePT, BOT_CSW_STATUS_Enum CSWstatus);

此函数用于处理错误状态，包括端点 STALL 和发送 CSW。

小型计算机系统接口 (SCSI) 命令

小型计算机系统接口 (SCSI) 命令为 SCSI 设备和主机之间提供有效的通信。大容量存储的范例代码实现了一部分 SBC-3 和 SPC-4 命令。更多信息请参考 SBC-3 和 SPC-4 规格。下表显示了大容量存储范例代码所支持的命令。这些命令在“usb_scsi_block_command.c”文件实现。BOT 层将传送命令块 (CB) 给 SCSI 层的 SBC_CMDHandler() 函数。该函数查找命令的操作码，并执行 SCSI 命令的相应函数。

表 19 大容量存储所支持的 SCSI 命令集

命令名称	操作码	描述
TEST UNIT READY	0x00	检查逻辑单元是否准备就绪。
REQUEST SENSE	0x03	设备服务器发送参数数据 (包括传感器数据) 到应用客户端。
INQUIRY	0x12	向应用客户端发送逻辑单元和 SCSI 目标设备信息。
MODE SENSE (6)	0x1A	向应用客户端报告参数。
READ FORMAT CAPACITY	0x23	主机请求报告当前媒体的可格式化能力。
PREVENT ALLOW MEDIUM REMOVAL	0x1E	请求逻辑单元使能或除能移除媒体。
READ CAPACITY (10)	0x25	此命令请求设备服务器传送 8 个字节的参数数据到数据输入缓冲区，该数据描述直接访问块设备的容量和媒体格式。
READ (10)	0x28	此命令请求设备服务器读取指定的逻辑块，并将它们传送到数据输入缓冲区。

命令名称	操作码	描述
WRITE (10)	0x2A	此命令请求设备服务器从数据输出缓冲区传送指定的逻辑块，并将它们写入媒体。

媒体访问

媒体访问的相关函数位于“ram_disk.c”或“sd_disk.c”文件中。BOT 层提供“BOT_Media_TypeDef”结构用于媒体信息。磁盘信息和访问函数被该结构定义为全局变量，“usb_bulk_inly_transport.c”文件中的“gMediaInfo”。下表显示了“gMediaInfo”变量的成员。

表 20 “BOT_Media_TypeDef”结构成员

成员	描述
u32 uId;	媒体唯一的 ID。此 ID 用于识别不同媒体上支持多个 LUN 的单一媒体驱动器。SCSI 层会发送这个 ID 给媒体驱动器。
BOT_MediaSize_Typedef Size;	尺寸信息，包括块数，块大小和媒体大小。
pInit Init;	媒体的初始化函数。
pReadFun Read;	媒体的读函数。
pWriteFun Write;	媒体的写函数。
pGetStatusFun GetStatus;	SCSI 层使用此函数来测试媒体是否准备就绪。
u8 *pSBC_InquiryData;	查询命令的数据指针。
u8 *pSBC_SenseData;	请求检测数据命令的数据指针。

用户应当按照上面的格式执行其他媒体的访问函数。BOT 和 SCSI 层通过上述信息和函数访问特定的媒介。下表给出了 RAM 磁盘访问函数的例子。

表 21 RAM 磁盘访问函数范例

函数	描述
u32 RAMDISK_Read(u32 uId, u8 **pBuffer, u32 uAddress, u32 uLength);	读媒体参数。SCSI 读命令使用此函数读媒体。
u32 RAMDISK_Write(u32 uId, u8 *pBuffer, u32 uAddress, u32 uLength);	写媒体参数。SCSI 写命令使用此函数写媒体。
u32 RAMDISK_GetStatus(u32 uId, BOT_MediaSize_Typedef *pSizeInfo);	此函数用于报告媒体状态。SCSI 的“TEST UNIT READY”命令使用此函数来测试媒体状态。

端点配置

大容量存储范例代码使用三个端点用于控制和批量传输。下表显示了端点的数量、方向和传输类型。端点的详细设置，可以在“ht32xxxx_usbdconf.h”文件中找到。

表 22 端点的 HID 范例代码

端点号	传输类型
端点 0	控制 IN 或 OUT
端点 1	批量 IN
端点 2	批量 OUT

- 端点 0（控制）：用于 USB 标准/类请求。
- 端点 1（批量 IN）：用于大容量存储 BOT 协议的 CSW 传输。
- 端点 2（批量 OUT）：用于大容量存储 BOT 协议的 CBW 传输。

类请求

大容量存储 BOT 协议定义了两个类专用请求。这些类请求可以在大容量存储范例代码的“ht32_usbd_class.c”文件中执行。下表显示了所有已定义的类请求及其描述。

表 23 大容量存储 BOT 的类请求

类请求	描述
仅批量大容量存储复位	该请求用于复位大容量存储设备及其相应接口。
获取最大 LUN	该请求用于确定设备所支持的逻辑单元数量。LUN 应从 LUN 0 开始到 15（最大）连续编号。

配置

范例代码支持多个 LUN（逻辑单元号）功能，通过单个 USB 大容量存储设备可访问不同的媒体。此范例代码能够支持 RAM 磁盘和 SD 卡。RAM 磁盘和 SD 卡功能由“usb_bulk_only_transport.h”文件使能或除能。同一个文件的 LUN 数量也可以改变。

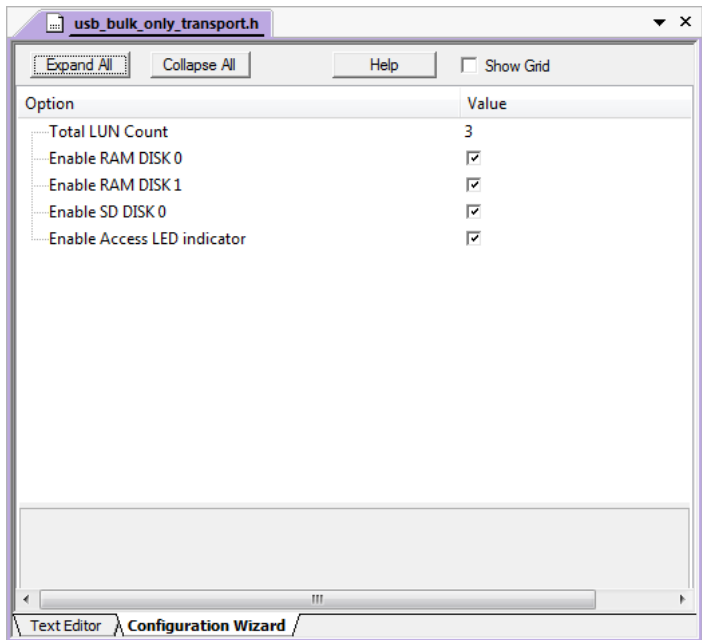


图 17 总 LUN、媒体和访问 LED 的配置

RAM 磁盘的 FAT12 范例数据显示了 PC 主机上的磁盘内容和信息。FAT 表被定义在“ram_disk.c”文件中名为“guMemory0”和“guMemory1”的变量中。文件名、文件内容、磁盘标签等都可以在 FAT 表中找到和修改。更多信息请参考 FAT12 规格。

USB视频

简介

此章节利用 HT32 系列开发板展示 USB 视频设备范例。送出 320x240 单色变化影像至 PC 显示，展示 USB 等时传输功能。USB 视频类 (UVC) 装置为 USB 设备类所定义的通用视频影像装置，包含动态影像装置，如数字摄影机、摄像头等装置。此装置连接至 PC，不需要自行撰写任何驱动程序，即插即用。

使用USB视频范例代码

此范例使用HT32 系列开发板并将其撰写为USB UVC Device。利用PC上通用影像程序开启装置。UVC装置即会传送影像数据至PC，完成等时数据传输功能。连接至PC后可在设备管理器下看到装置，如图 18。通过任何 PC 端的影像程序软件开启，例如 AMCAP (<http://noeld.com/programs.asp?cat=video>)。以AMCAP为例，在PC端安装完此影像程序软件。开启此软件，在Device中，选择USB-UVC DEMO。并点选Option→Preview，流程如图 19。在此范例中，Preview勾选后，USB即开始上传数据，传输单色变化影像结果如图 20。

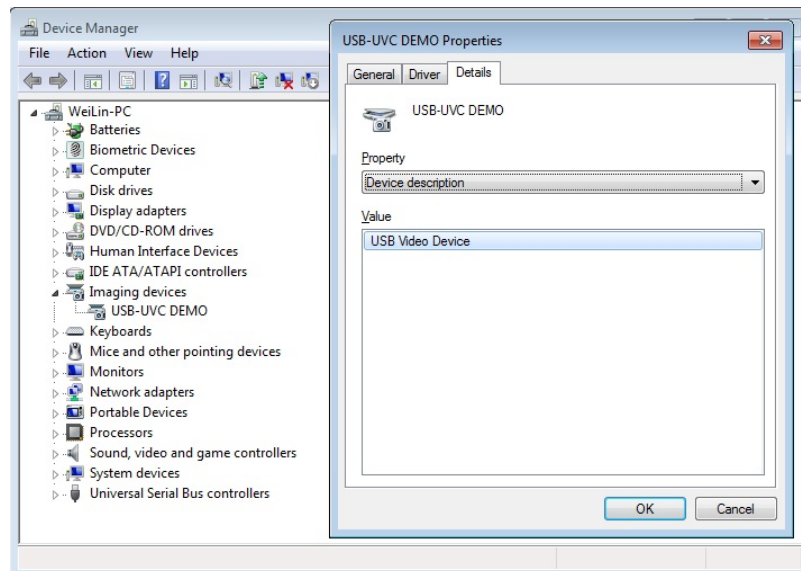


图 18 设备管理器的 USB 视频设备

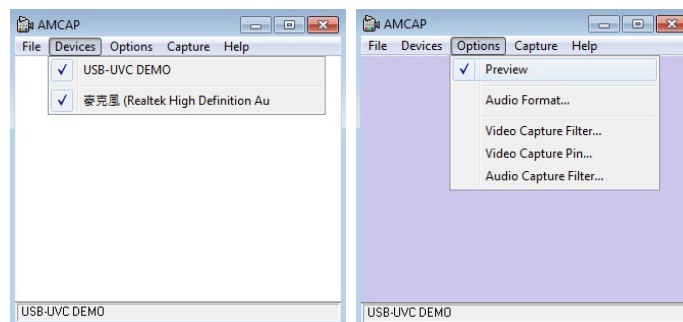


图 19 AMCAP 中 USB 视频设备选择

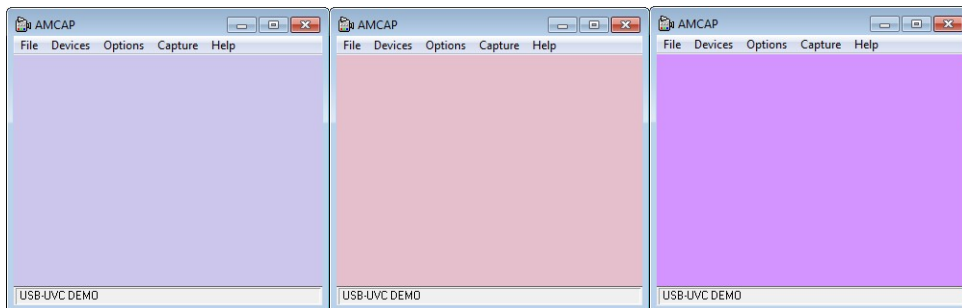


图 20 USB 视频设备单色变化影像

USB 视频类的实现

USB 视频范例程序采用 USB 视频设备类规格中定义影像格式为未压缩的有效载荷挂载 UVC 装置，并实现等时数据传输。关于更多影像格式定义请参考“USB Video Payload”相关规格。

端点配置

UVC 设备使用三个端点进行数据传输，如表 24。

表 24 USB 视频范例代码的端点

端点号	传输类型
端点 0	控制 IN 或 OUT
端点 1	IN 中断
端点 4	等时 IN

- 端点 0（控制）：USB 枚举数据传输和响应 USB 请求。
- 端点 1（中断）：中断端点用于状态返回。此端点是可选的，但在一定条件下可能是强制的。
- 端点 4（等时）：主要作为影像数据传输，将有效载荷头及有效载荷数据传送至 PC。

类请求

为了实现 UVC 装置，设备需支持 USB 视频类中所定义的 USB 类请求。如表 25。这些函数定义在“ht32_usb_class.c”文件中，如需更详细了解，请参考“USB Device Class Definition for Video Devices”规格书第 4 章。

表 25 UVC 类请求

类请求	描述
GET_CUR	返回当前状态。
GET_MIN	返回最小值
GET_MAX	返回最大值
GET_RES	返回分辨率
GET_DEF	返回默认值
GET_LEN	返回长度
GET_INFO	查询功能和状态
SET_CUR	设置当前状态

配置

用户可针对 UVC 的范例程序进行修改。此范例程序提供用户常用变量以及等时数据传输功能，方便用户依照需求进行修改。

- 变更影像尺寸大小，如表 26:

表 26 变更影像尺寸

文件名	定义名称	描述
ht32_usbd_class.h	DESC_IMAGE_WIDTH	影像宽度
	DESC_IMAGE_HEIGHT	影像高度

- 等时数据传输功能

```
void USBDClass_Endpoint4(USBD_EPTn_Enum EPTn)
```

用户自行将需要传输数据的程序内容撰写于此，即可轻易实现通过 USB 等时传输数据。此函数位于 ht32_usbd_class.c 文件中。

- USB 相关硬件配置

使用者可以在 ht32fxxxx_usbdconf.h 文件中修改 USB 相关配置，如图 21。另外，USB1.1 规范中，定义等时传输速率最大为 1023 字节/ms，而 UVC 装置在 1ms 中所能传送的数据大小为端点缓冲区所设定长度，使用者在修改上需特别注意。

Endpoint1 Configuration	<input checked="" type="checkbox"/>
Endpoint Address (EPADR)	1
Endpoint Enable (EPEN)	<input checked="" type="checkbox"/>
Endpoint Transfer Type	Interrupt
Endpoint Direction (EPDIR)	IN
Endpoint Buffer Length (EPLN) (in byte)	8
Endpoint Interrupt Enable Settings (EPIER)	
Endpoint2 Configuration	<input type="checkbox"/>
Endpoint3 Configuration	<input type="checkbox"/>
Endpoint4 Configuration	<input checked="" type="checkbox"/>
Endpoint Address (EPADR)	4
Endpoint Enable (EPEN)	<input checked="" type="checkbox"/>
Endpoint Transfer Type	Isochronous
Endpoint Direction (EPDIR)	IN
Endpoint Buffer Length (EPLN) (in byte)	500
Single/Double Buffer Selection (SDBS)	Single Buffer
Endpoint Interrupt Enable Settings (EPIER)	

图 21 USB 端点的配置 (ht32fxxxx_usbdconf.h)

虚拟COM端口

简介

此章节利用 HT32 系列开发板展示 USB 虚拟 COM 端口（VCP）范例。执行简单的数据传输 loopback 行为，展示 USB 批量 IN/OUT 功能。USB 是连接计算机系统与周边装置的标准技术规范，但在电脑刚问世时，外接式装置的传输大多通过 COM 端口传输，现今仍有许多软件与周边使用。USB 提供标准的类以完成 COM 端口功能，大幅减少软件重新开发的时间，并且提高传输速度。USB VCP Demo 提供 HT32 系列 IC 的固件范例以及 PC 驱动器。以下章节将会说明如何使用此范例。

使用虚拟COM端口范例代码

此范例使用 HT32 系列开发板并将其撰写为 USB VCP Class Device。在 PC 上使用 Hyper Terminal 传送数据至 USB VCP Class Device（HT32 系列开发板），HT32 系列开发板接收到 USB 数据并回传至 Hyper Terminal，完成数据 Loopback 行为。发展环境示意如图 22。

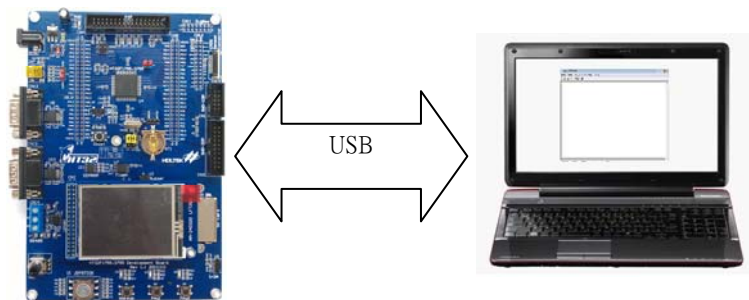


图 22 虚拟 COM 端口范例代码的发展环境

在第一次执行 USB VCP Demo 时，将 HT32 系列开发板与 PC 连接时需要安装程序，此时搜寻 USB VCP Demo 程序中的 HT32_VCP.inf 并且安装。安装完成后，在设备管理器中可以看到新增的 Holtek USB 虚拟 COM 端口，代表安装完成，如图所示。

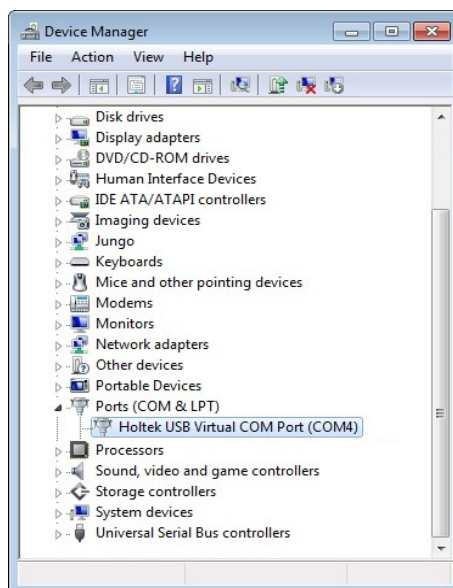


图 23 设备管理器的 VCP 设备

虚拟COM端口类的实现

VCP 范例程序使用 USB 通信设备类（CDC）规格中定义抽象控制模式实现通用的 COM 端口设备。

端点配置

虚拟 COM 端口设备使用四个端点进行数据传输。

- 端点 0（控制 IN/OUT）：USB 枚举数据传输和响应 USB 标准请求。
- 端点 2（IN 中断）：主要传送通知请求，在此范例中并无传输任何数据。
- 端点 3（批量 OUT）：接收 PC 送出 COM 端口数据。
- 端点 1（批量 IN）：传送 COM 端口数据至 PC。

类请求

在 USB 通信设备类（CDC）的抽象控制模式中定义虚拟 COM 端口设备必要的特殊类请求。

- SET_LINE_CODING：此命令用于配置串口参数，包含比特率、停止位的数目、奇偶校验和数据位数。
- GET_LINE_CODING：此命令用于请求串口参数，包含比特率、停止位的数目、奇偶校验和数据位数。
- SET_CONTROL_LINE_STATE：此命令用于设定 RS-232 信号 RTS 和 DTR。